Jurnal Teknologi Maklumat & Multimedia **3**(2006): 139-155

Application of Simulated Annealing and Genetic Algorithms in Solving Single Level Lot-Sizing Problems

NASARUDDIN ZENON, ROSMAH ALI & AB RAHMAN AHMAD

ABSTRACT

The single level lot-sizing problem arises whenever a manufacturing company wishes to translate an aggregate plan for production of an end item into a detailed planning of its production. Although this problem is widely studied in the literature, only laborious dynamic programming approaches are known to guarantee global minimum. Thus, stochastically-based heuristics that have the mechanism to escape from local minimum are needed. Two implementations of stochastic local search techniques for solving single level lot-sizing problems are proposed and the results of applying them to example problems are discussed. In the first implementation, simulated annealing is used to examine the neighborhoods of the replenishment points of an initial schedule obtained by following the standard Silver-Meal criterion. This provides a way of escaping local minimum in the total relevant costs per unit time for the current replenishment and thus improves the initial lot-sizing schedule. In the second implementation, a lot-sizing population-generating heuristic is used to feed chromosomes to a genetic algorithm which will try to find the optimal lot-sizing scheme without going through the process of examining the legality of the scheme for every generation. The application of the heuristic to generate an initial population results in a faster convergence in finding the optimal lot-sizing scheme. The result of this research shows that for uncapacitated lot-sizing problems using the sample of data, simulated annealing outperforms genetic algorithm in terms of run time and convergence rate. However, genetic algorithm outperforms simulated annealing in terms of a lower total production cost for problems with production penalties.

Keywords: Simulated annealing; genetic algorithm; lot-sizing

ABSTRAK

Masalah pensaizan lot satu-aras timbul apabila sebuah syarikat pembuatan ingin menterjemahkan pelan pengeluaran agregat untuk suatu item akhir *kepada pelan pengeluaran yang terperinci.Walaupun masalah ini kerap dikaji, namun setakat ini hanya pendekatan pengaturcaraan dinamik yang rumit dapat menjamin nilai minimum global. Oleh itu, heuristik stokastik yang mempunyai mekanisma untuk melepasi minimum global adalah diperlukan. Dua perlaksanaan teknik carian tempatan stokastik bagi menyelesaikan masalah pensaizan lot satu-aras dicadangkan, dan hasil perlaksanaan ini dibincangkan. Dalam perlaksanaan pertama, simulated annealing digunakan untuk memeriksa kejiranan titik-titik pembaharuan semula bagi jadual awal yang diperolehi melalui kaedah Silver Meal. Ini memberikan satu cara untuk melepasi minimum tempatan dalam nilai kos keseluruhan seunit masa untuk pembaharuan semasa dan memperbaiki jadual pensaizan lot awal. Dalam perlaksanaan kedua, suatu heuristik yang menjanakan populasi untuk pensaizan lot digunakan untuk menyediakan kromosom bagi algoritma genetik yang akan mendapatkan skema pensaizan lot optimum tanpa memeriksa kesahihan skema bagi setiap generasi. Penggunaan heuristik untuk menjanakan populasi awal menghasilkan penumpuan yang lebih cepat dalam pencarian skema pensaizan lot yang optimum. Hasil penyelidikan ini menunjukkan bahawa untuk data sampel bagi masalah pensaizan lot kapasiti-tanpa-batas, simulated annealing mengatasi algoritma genetik dari aspek masa larian dan kadar penumpuan. Walau bagaimanapun algoritma genetik mengatasi simulated annealing daripada aspek jumlah kos keseluruhan pengeluaran yang lebih rendah bagi masalah dengan penalti pengeluaran.*

Kata kunci: Simulated annealing; algoritma genetik; pensaizan lot

INTRODUCTION

In manufacturing environment, a lot size refers to the amount of a particular item that is ordered from the plant or issued as a standard quantity to the production process. Lot-sizing or lot size scheduling refers to the determination of appropriate lot sizes of items to be produced in each period of the production planning horizon such that the setup and the inventory holding costs associated with the schedule for the whole of the planning horizon are minimized. The single level lot-sizing problem without backlogging is to find a feasible production schedule of end items over a time horizon consisting of T period such that the total inventory holding cost plus the setup cost is minimized. Assumptions made in this problem are that the initial inventory is zero and the first period demand is non-zero. Let d_t, p_t, and I_t be the demand rate, production quantity and inventory level at the end of period t respectively for $t = 1,2,\ldots, T$. Furthermore let C be the total variable cost which is the sum of the setup cost (S) plus the unit holding cost (h). The mathematical formulation of the problem can be stated as follows:

Minimize:

$$C = \sum_{T=1}^{T} \left[S\delta(p_t) + hI_t \right] \tag{1}$$

Subject to:

$$I_{t-1} + p_t - I_t = d_t \qquad (t = 1,2,\ldots,T)$$

$$I_0 = 0$$

$$p_t, I_t \geq 0 \qquad (t = 1,2,\ldots,T)$$

Where:

$$\delta(p_t) = \begin{cases} 0 \; if \;\; p_t = 0 \\ 1 \; if \;\; p_t > 0 \end{cases}$$

The lot sizes are simply the accumulated demands for each order interval and thus equal to $\sum_{t=c}^{e} p_t$ where $1 \leq c \leq e \leq T$. Item deliveries are planned only for periods with positive demands. If the demand in an order receipt period is zero, the order receipt is moved ahead to the first subsequent period with a positive requirement (Tersine 1994).

The Wagner-Whitin algorithm (WWA) (Wagner & Whitin 1958) and its variant, which are dynamic programming approaches in solving the single level lot-sizing problem as described previously, are the only known algorithms that will guarantee convergence to the optimum solution of the lot-sizing problem. However, the algorithms are often criticized as being difficult to explain and compute. For this reason, WWA often serves as a benchmark against which to measure the performance of non-optimal but less complex lot-sizing approaches (Tersine 1994).

The Silver-Meal (SM) heuristic (Silver & Meal 1973) uses criterion (2) to determine production quantities x_t for $t = 1,2,\ldots,T$. Specifically, given d_t, the demand in period t, the setup cost S and the holding cost per unit per time h, it is favorable to include d_k in the production x_t if,

$$\left(S + h\sum_{t=1}^{k} (t-1)d_t \right) / \, k \leq \left(S + h\sum_{t=1}^{k-1} (t-1)d_t \right) / \, (k-1) \tag{2}$$

The Silver-Meal heuristic, as shown by Zenon and Ahmad (2001) and Ritchie and Tsado (1986), is one of the most efficient techniques with reasonable cost performance (with respect to equation (1)) for lot-sizing problems having deterministic time-varying demand requirements. However, as the variation in demand increases, as reflected in higher values of coefficient of variation in demand (CVD), the performance of the algorithm deteriorates (please refer to Zenon Ahmad (2001) and Ritchie Tsado (1986) for detailed results). Indeed, one of the originators of the algorithm in Silver and Peterson (1985) noted that this method guarantees only a local minimum in the total relevant costs per unit time for the current replenishment and that there are two situations in which the algorithm can lead to significant cost penalties in equation (1). These are:

1. When the demand pattern drops rapidly with time over several periods.
2. When there are a large number of periods having no demand.

Despite the shortcoming of the SM heuristic in dealing with complex demand patterns, any criterion based on (2) is an excellent determinant for production quantities in any particular period. Thus, in this paper we examine the advantages of adapting two stochastic based methods for solving single level lot-sizing problems. In the first implementation, the SM criterion provides an initial schedule in the form of periodic production quantities, which we will call centre points, to be manipulated by simulated annealing (SA) to find better schedules. In the second implementation, genetic algorithms GA is fed with populations of valid lot size schedules generated by a heuristic developed in Ahmad, Zenon and Mi Yusuf (1999).

An Overview of Simulated Annealing

SA is a search process that has its origin in the field of statistical mechanics. It was first developed by Metropolis et al. (1953) as an approximate numerical simulation model for describing the physical annealing process of condensed matter. For optimization purposes, SA is a probabilistic method which operates on an energy function $E(X_i)$ of the independent variables X_i. An assignment for X_i is called a configuration. The procedure walks in the space of these configurations, where the direction of the next step is dependent on the energy value of the neighboring configurations (Beringer et al 1994). A neighbor to a configuration X_i differs only in the assignment of one variable.

In general, if t is a control parameter and the procedure is started from the maximum value of t, a move from configuration X_i to X_{i+1} is made with a probability $P_t(X_{i+1} \leftarrow X_i) = P_t(E(X_{i+1}) - E(X_i))$ for $0 < t < T$ where T is to be determined by the modeler as the maximum allowable value of t. As noted by Beringer et al (1994), in general SA has the tendency to select configurations with lower energy. Furthermore, it can be proved that SA always converges to a local minimum of the energy function (a stable state) as $t \rightarrow 0$ provided the transition probabilities P_t are chosen such that these distributions obey:

$$\frac{P_t(X_{i+1} \leftarrow X_i)}{P_t(X_{i+1} \leftarrow X_{i+1})} = \exp\left(-\frac{E(X_{i+1}) - E(X_i)}{t}\right)$$

Over the last two decades, SA has been applied to many scheduling problems, in academia as well as in industry, with considerable success (Pinedo & Chao 1999). For scheduling purposes, SA selects candidate schedules from a predefined neighborhood of a schedule in a manner that loosely mimics the gradual cooling of a metal. Early in the search process, a traditional SA may make wild random changes to the schedule. However, as time goes on, the schedule becomes less volatile (i.e., is "cooled") and the approach becomes more and more greedy (Hopp & Spearman 2000).

USING SA TO IMPROVE SM BASED LOT-SIZE SCHEDULES

We begin by searching for the lot-sizing sequence S_1 that minimizes C in (1) based on SM criterion. The algorithm then performs a number of iterations. If S_t is the current lot-sizing sequence and S_0 is the best sequence found so far at iteration t of the algorithm then $C(S_t) > C(S_0)$ where $C(S_t)$ and $C(S_0)$ denote the corresponding values of the objective function. The sequences S_t and S_0 are obtained by perturbing the initial sequence S_1 at the replenishment points previously obtained following the SM criterion.

The neighborhood design of S_t is as follows. Let s_{tr}, $r = 1 \ldots N$ be a set of N replenishment points in S_t obtained earlier following the SM criterion. Each of these points will be called a center point. Starting with $r = 1$, the neighborhood of each s_{tr} is constructed by shifting all productions corresponding to non-zero demands in period r+1, r+2,…, r+n, where n is the next center point, to period r, for $1 \le r \le n \le N \le T$. The total variable cost associated with the new sequence created is compared against the previous total variable cost after the production in r+n is shifted. If the new cost is less than the previous cost then the candidate sequence S_n will be assigned to the new sequence. Otherwise, production in r+n+1 is also shifted to period r and the above process is again repeated. If perturbation at center point n leads to a fruitless search for a lower cost then set $r = n$ and the neighborhood of the next center point will be examined in a similar way.

The algorithm, in its search for an optimal schedule, moves from one center point to another and the selection of a candidate schedule is done in a sequential way. If $C(S_n) < C(S_t)$, a move is made by setting $S_{t+1} = S_n$. Furthermore, if $C(S_n) < C(S_0)$, then S_0 is set equal to S_n. However, if $C(S_n) \ge C(S_t)$, a move is made to S_n with probability:

$$P(S_t \leftarrow S_n) = \exp\left(\frac{C(S_t) - C(Sn)}{\beta_t}\right) \tag{3}$$

Equation (3), which is based upon the Boltzmann distribution function, is also known as the Metropolis criterion. The schedule S_n is rejected in favor of the current schedule with probability 1 - $P(S_t \leftarrow S_n)$ by setting $S_{t+1} = S_t$. Temperatures $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_t \geq \ldots \geq 0$ are control parameters. These temperatures are gradually lowered throughout the algorithm from a sufficiently high starting value, *startTemp* = $\sqrt{12}$ / (2.0 * *randMax*), where *randMax* is the maximum possible value for the random number generator, to a "freezing" temperature, *stopTemp* = 1, where no further changes occur. Here, the temperature is decreased in stages according to the number of iterations *t*, using the temperature update equation β_{t+1} = *factor* $*\beta_t$ where *factor* = exp(1n(*stopTemp / startTemp*)/(*N*-1). At each stage, the temperature is kept constant until thermal quasi-equilibrium is reached. Quasi-equilibrium state is a condition when the system remains infinitesimally close to an equilibrium state at all times.

The whole process of determining the initial temperature, the stopping temperature, the temperature decrement formula between successive stages and the number of transitions for each temperature value is called the cooling schedule. After repeated experimentations we found that a cooling schedule adapted from the works of Masters (1993) as described previously leads to faster convergence. Masters uses simulated annealing to escape from local minima frequently encountered in training feed-forward artificial neural networks using the back-propagation algorithm.

Several stopping criteria are possible for this procedure. However the one we have chosen is to let the procedure run until $r = N$, i.e. until the entire center points are perturbed. Algorithm 1 summarizes the procedure.

Algorithm 1. Simulated Annealing extension of SM heuristic

```
Set $t,r \leftarrow 1$.
Set $\beta_1 \leftarrow$ startTemp
Select an initial lot-sizing sequence $S_t$ based on SM criterion
Set $S_0 \leftarrow S_t$
While $(r \leq N)$ {        // iteratively select center points
    Select a candidate schedule $S_n$ from the neighborhood of $S_t$ as defined
earlier.
If $C(S_n) < C(S_t)$,
    Set $S_{t+1} \leftarrow S_n$
    If $C(S_n) < C(S_0)$,
        Set $S_0 \leftarrow S_n$
    Else if $C(S_n) \geq C(S_t)$,
        Set $U_t \leftarrow$ random_number $\in$ Uniform (0,1)
        If , $U_t \leftarrow P(S_t \leftarrow S_n)$            // Metropolis criterion
```

```
            Set $S_{t+1} \leftarrow S_n$
            Skip the next center point
        Else
            Set $S_{t+1} \leftarrow S_t$
            Move to the next center point
    Set $\beta_{t+1} \leftarrow factor * \beta_t$
    Set $t \leftarrow t + 1$
    Set $r \leftarrow r + 1$
                                                    // end while
```

A GA IMPLEMENTATION

The working principle of a GA can be depicted in Figure 1. The main part of a GA cycle constitutes of artificial genetic operators given the names mimicking their biological counterparts. These operators are called reproduction, crossover and mutation. A GA begins its search with a random set of solutions, instead of one solution as it is normally done in classical search and optimization methods. The random set of solutions constitutes a generation of population.

The fitness function value of a solution is a metric that measures a relative merit of the solution based on an objective function (for a single-objective optimization problem with constraints) such as given in (1) and constraint functions. For each generation, the genetic operators will be applied to the generation if a termination criterion is not met.

The process of identifying good (user defined or simply above average) solutions in a population and eliminating bad solutions by replacing them with multiple copies of good solutions while maintaining a constant population size are basically the function of the reproduction operator. Obviously as noted by Deb (2001), by making more copies of good solutions at the expense of not-so-good solutions, the reproduction operator cannot create any new solution in the population. Therefore more operators are needed to create new solutions, and they are namely the crossover and the mutation operators.

There are a number of possible crossover operators in the GA literature. For detailed descriptions of the designs of the operators the readers are referred to the seminal works on GA such as Holland (1975) and Goldberg (1989) and the more recent works of Spears (1998) and Deb (2001). However, if we were to represent fitness function values as binary strings for example, we can simply illustrates the crossover operator as an operator that select two good solutions (called parent solutions) at random from the ones previously selected by the reproduction operator. Let us define a single-point crossover operator that proceeded to choose a cross site (again at random) along the binary string length so that the contents of the right side of this cross site can be exchanged between the two strings. Thus the operator has just created two new strings called offspring.

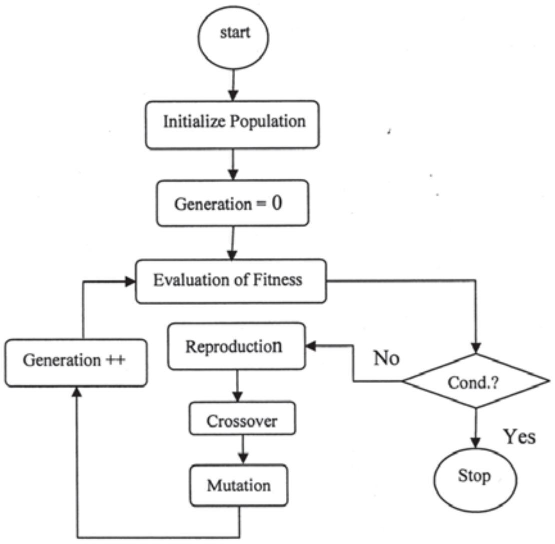

FIGURE 1. A flowchart of a GA process

The expected fitness values of the offspring are usually better than the parent because prior to the crossover process the reproduction operator had already reproduced parents with some good bit combinations in their string representations (refer to Goldberg (1989).

The crossover operator is mainly responsible for the search aspect of GA, even though the mutation operator is also used for this purpose (Deb 2001). The mutation operator is a bit-wise manipulator that changes a 1 to a 0 and vice versa, with a mutation probability of p_m. The significance of the mutation operator is that it can reduce the chance of the search from being entrapped in a neighborhood of a local optimal point.

A basic problem usually encountered after the crossover process as far as lot-sizing is concerned, is the generation of children that does not preserve the accumulation of demands at a non-zero demand point and thus producing invalid lot-size schedules. Thus a method is required to generate an initial valid population of lot size schedules with respect to the Master Production Schedule (MPS). Secondly, an effective crossover scheme is required to preserve schedules validity throughout the crossover process. The generation of valid lot size schedules is accomplished with the use of a heuristic as described below, and a description of the crossover scheme that produces valid schedules follows after that.

As said in the introduction our assumption is that each demand vector d_t has discrete values that came from fixed horizon environments. The basic idea of the lot-sizing population-generating heuristic is to start shifting productions backward in time from the end of the horizon systematically while retaining enough inventories to satisfy demands in the later period. This backward shifting of productions will create alternate or subsequent periods with zero production levels thus minimizing the overall setup cost. For problems involving variable setup and holding costs, the algorithm will advertently create enough zero production levels thus avoiding periods with large setup and/or holding costs. With this idea in mind, the heuristic is presented as Algorithm 2.

Algorithm 2: Population-Generating Heuristic

```
For j=1 to T
    a(j) = d(j)     /* The initial demand vector from MPS */
While (j ≤ T – 3) do     /* The number of times backshift is performed is T-
3 times */
Step 1          /* Backward shifting of production quantities $a_j$ */
    While (i > posAllele) do
        If a(i-posAllele) > 0
            For t = (i-(posAllele-1) t o i
        If a(t) ≠ 0
                   a(i-posAllele) = a(i-posAllele) + a(t)
                   a(t) = 0
            End If
          End For
        End If
    End While

Step 2   /*  Assignment of $a_j$ to chromosome vector $pop_1(j)$. If period 1 and
period 2 have non-zero lot-sizes, generate 1 more chromosome $pop_2(j)$ */

    For j=1 to T
    pop1(j) = a(j)
            If (k=1) OR (i ≤ posAllele) AND pop1(2) ȷ  0
           pop2(1) = a(1) + pop1(2)
           pop2(2) = 0
           for j=3 to T
            pop2(j) = pop1(j)
    Else
        inheritParent = inheritParent + 1
    End If
Step 3   /* Create an exception for the case when T is divisible by the
number of shifting steps.   */
           If (T % (posAllele +1) = 0
                For j=1 to T
                        pop2(j) = 0
                Else
inheritParent = inheritParent + 1
     End If
                  If (inheritParent = 2)
              For j=1 to T
            pop2(j) = d(j)
              inheritParent=0;
         End If

Step 4   /* dual shift processes */
/* For every chromosome pop_k(j) in Step 1, dual shift processes are generated
by shifting productions a_j only to the left of the period h = T/2 + 1.The dual
shift process is similar to the above 3 steps except for the following
replacements: */
        i  ← h = T/2 + 1
      pop1(j) ← pop3(j)
      pop2(j) ← pop4(j)
/*For an exception case when T is divisible by the number of shifting steps,
a replacement T ← T/2 + 1 is required  for testing the divisibility condition.
*/
End While
```

posAllele is the position of the allele or feature value in the chromosome. *Step 2* and *Step 3* can be combined in a single step. However, for the sake of readability, the steps are separated in the algorithm given. *Step 4* is necessary in order to simulate lot size schedules with accumulations of production at least at two periods, one in the beginning and one in the middle of horizon. The main advantage of performing this step is the availability of more useful chromosomes with non-zero crossover points.

The population-generating heuristic generates demand sequence matrices such as listed in Appendix A and Appendix B which encode valid demand requirements for all periods in the planning horizon. Pop(0, j) for 1£ j £ T represents the original demand rate for each period j in the MPS. If we let M be the total number of chromosomes generated by the lot-sizing population-generating heuristic, then pop(i, j) represents the lot-sizing schedules in each chromosome i = 1, 2,…, M. Each pop(i, j) will have a specific fitness value calculated using Equation 1.

We used a 1-point crossover scheme at a locus (string position) having a non-zero allele (feature value) in both parents. This way, the children generated will preserve the accumulation of demands at a non-zero demand point and thus producing valid schedules as required by the characteristic-preserving criteria discussed in Kobayashi, Ono and Yamamura (1995; 1996). As an additional note, the probability of crossover $p_c = 1.0$ and the probability of mutation, $p_m = 0.001$ for MPS requirements with $T \leq 30$; also been used.

NUMERICAL EXAMPLES

All the results obtained in this section are based on applying several examples to SA and GA developed using Java 2 on a Pentium III processor with a clock speed of 450 MHz, with 128MB of memory, and running Windows 98.

Three datasets with various cost structures were used to test SA and GA. Each dataset contains a demand vector D_j for $1 \leq j \leq T$ where $T \leq 30$ in all cases. The three datasets are given in Table 1. Table 2. and Table 3. respectively.

Dataset 1 contains 2 periods with zero demand whereas each period in Dataset 2 has non-zero demand. For Dataset 1 we used a constant setup cost of 100 for each period and a holding cost of 1. On the other hand, Dataset 2 imposes penalties in certain periods by way of large setup (S_j) and holding

TABLE 1. Dataset 1

Period j	1	2	3	4	5	6
D_j	75	0	33	28	0	10

TABLE 2. Dataset 2

Period j	1	2	3	4	5	6
D_j	10	15	7	20	13	25
S_j	20	17	10	20	5	50
h_j	1	1	1	3	1	1

TABLE 3. Dataset 3: Production request for 30 periods

Period j	Demand D_j	Period j	Demand D_j	Period j	Demand D_j
1	81	11	50	21	77
2	67	12	47	22	96
3	53	13	7	23	64
4	96	14	88	24	87
5	35	15	20	25	51
6	65	16	25	26	7
7	27	17	88	27	85
8	81	18	74	28	82
9	84	19	62	29	53
10	32	20	52	30	96

(h_j) costs. We compared the feasible lot-sizing schedules produced by SA and GA with optimal and near optimal schedules produced by other algorithms. Table 4. and Table 5. summarize the results when SA and GA are compared with WWA, SM and LUC algorithms.

The feasible production schedules for Dataset 1 and Dataset 2 that were generated using the lot-sizing population-generating heuristic are given in Appendix A and Appendix B respectively. The feasible production schedules for data in Table 4. that were generated by the same heuristic are available upon request from the first author. As shown in Table 5. pop(4,j) for $1 \leq j \leq 6$ listed in Appendix A is the required schedule to minimize the total variable cost. This shows that the population-generating heuristic is able to produce near-optimal schedule even prior to a crossover to be carried out by GA.

The results from applying GA to Dataset 2 showed the superiority of GA compared to SM and LUC heuristics in producing schedules involving variable setup and holding costs. The last chromosome produced by GA has a fitness value representing a total cost that is less than 1% from the optimal WWA.

TABLE 4. Comparative results for Dataset 1

	Period j	1	2	3	4	5	6	Total Variable
	D_j	75	0	33	28	0	10	Cost, C
	WWA	75	0	71	0	0	0	258
Production	SM	75	0	71	0	0	0	258
Quantity	LUC	75	0	61	0	0	10	328
	SA	75	0	71	0	0	0	258
	GA	75	0	71	0	0	0	258

TABLE 5. Comparative results for Dataset 2

	Period j	1	2	3	4	5	6	Total Variable
	D_j	10	15	7	20	13	25	Cost, C
	WWA	10	22	0	20	38	0	94
Production	SM	32	0	0	20	13	25	124
Quantity	LUC	32	0	0	33	0	25	158
	SA	32	0	0	20	38	0	99
	GA	25	0	27	0	38	0	95

Performance of GA in terms of convergence rate utilizing the parameters is measured using demand data listed in Table 3. Two cost structures are used. The setup and the holding costs for the first cost structure is S = RM2.6

and h = RM2.39 respectively. The second cost structure has S = RM300 and h = RM0.2. While the first cost structure has an S/h ratio circa 1, thus allowing a lot of periods to have positive production quantities, the second cost structure on the other hand will force the algorithm to produce schedules with sparse production periods due to the very high setup cost relative to the holding cost.

The result of applying GA to a population generated by the population-generating heuristic is compared to the result obtained by SA. The optimum schedule for the first cost structure is known to be a lot-for-lot assignment because of the small difference in the setup and holding costs.

The convergence pattern of GA to the optimum total cost is compared to the result produced by SA and is depicted in Figure 2. and Figure 3. for the first and the second cost structures respectively. Table 6. and Table 7. summarize the result of applying GA and SA to demand data in Table 3. using the two cost structures. Table 8. lists the optimum production lot-size schedule with a total production cost of 2312.20. The total cost obtained is confirmed to be optimum by feeding the data to WWA.

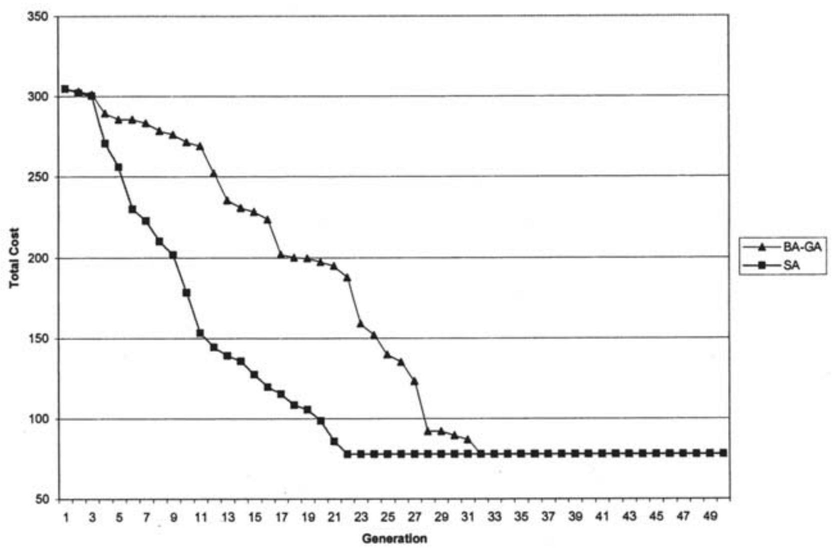

FIGURE 2. Convergence of GA and SA for the first cost structure

CONCLUSION

Clearly, for the uncapacitated lot-sizing example above, SA outperforms GA in terms of run time and convergence rate. This is due to the fact that our implementation of SA is designed to examine the neighborhoods of at most

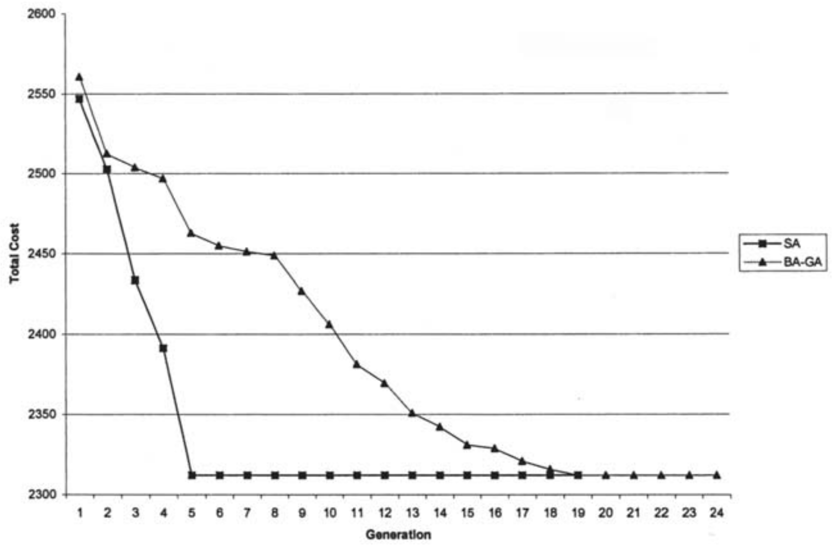

FIGURE 3. Convergence of GA and SA for the second cost structure

TABLE 6. Comparative results for Dataset 3 with the First Cost Structure

	GA	SA
Generation	50	-
Population	74	30 centers (max)
Optimum Total Cost	78.00	78.00
Lot-size Schedule	Lot-for-lot	Lot-for-lot
Start of Optimum Generations	32	22 (iteration)
Run Time	118000ms	2420 ms

TABLE 7. Comparative results for Dataset 3 with the Second Cost Structure

	GA	SA
Generation	25	-
Population	74	5 centers
Optimum Total Cost	2312.20	2312.20
Lot-size Schedule	Table 5.8	Table 5.8
Start of Optimum Generations	20	6 (iteration)
Run Time	12530ms	250 ms

TABLE 8. The optimum lot-size schedule produced by BA-GA and SA

Period	1	8	14	21	27
Production	424	301	409	382	316

30 center points corresponding to the 30-period planning horizon. GA on the other hand requires a large initial population size so that the randomized crossover and mutation operations can produce good reproduction candidates. However, the performance of GA for problems with constraints in terms of production penalties in some periods is encouraging. GA outperforms SM, LUC and SA in terms of achieving a lower total production cost. The population-generating heuristic is able to produce near optimum schedules as candidates for reproduction at a very early stage thus enabling GA to selectively work on ‘good’ chromosomes.

The encouraging results obtained lead us to the conclusion that our implementation of SA and GA for single level lot-sizing problems with or without production penalties can be extended for the capacitated multi-item lot-sizing problems. The problem of finding an optimal lot size schedule for a multi-item capacitated version of the problem is of special importance though it is known to be NP-complete. The suitability of stochastic searching techniques in this case gives us the opportunity to explore the possibility of embedding good lot-sizing criteria into stochastically based heuristics.

APPENDIX A

Chromosomes generated by Population-Generating Heuristic for Dataset 1

```
pop[0][j]: 75.0, 0.0, 33.0, 28.0, 0.0, 10.0
pop[1][j]: 75.0   0.0   61.0   0.0   0.0   10.0
pop[2][j]: 75.0   0.0   61.0   0.0   0.0   10.0
pop[3][j]: 108.0   0.0   0.0   38.0   0.0   0.0
pop[4][j]: 75.0   0.0   71.0   0.0   0.0   0.0
pop[5][j]: 75.0   0.0   33.0   28.0   0.0   10.0
pop[6][j]: 136.0   0.0   0.0   0.0   0.0   10.0
```

APPENDIX B

Chromosomes generated by Population-Generating Heuristic for Dataset 2

```
pop[0][j]: 10.0, 15.0, 7.0, 20.0, 13.0, 25.0
pop[1][j]: 25.0   0.0   27.0   0.0   38.0   0.0
pop[2][j]: 25.0   0.0   27.0   0.0   13.0   25.0
pop[3][j]: 32.0   0.0   0.0   58.0   0.0   0.0
pop[4][j]: 10.0   42.0   0.0   0.0   13.0   25.0
pop[5][j]: 10.0   15.0   7.0   20.0   13.0   25.0
pop[6][j]: 10.0   15.0   65.0   0.0   0.0   0.0
pop[7][j]: 25.0   0.0   65.0   0.0   0.0   0.0
pop[8][j]: 52.0   0.0   0.0   0.0   13.0   25.0
```

REFERENCES

Ahmad, A. R., Zenon, N. & Mi Yusuf, L. 1999. A Method For MRP Lot-sizing Problem Representation in Genetic Algorithm. *18th IASTED International Conference on Applied Informatics (AI 2000)*. 307-4-002.

Beringer, A., Aschemann, G., Hoos, H., Metzger, M. & Weib, A. 1994. GSAT and simulated annealing – A comparison. *Intellektik,* AIDA-94-01: 1-18.

Deb, K. 2001. *Multi-objective optimization using evolutionary algorithms.* Chichester: Wiley.

Goldberg, D. E. 1989. *Genetic algorithms for search, uptimization and machine learning*. Reading, MA: Addison-Wesley.

Holland, J. H. 1975. *Adaptation in natural and artificial systems.* Ann Arbor, MI: MIT Press.

Hopp, W. J., Spearman, M. L. 2000. *Factory physics: foundations of manufacturing management.* Singapore: McGraw-Hill International Edition.

Kobayashi, S., Ono, I. & Yamamura, M. 1995. An efficient genetic algorithm for job shop scheduling problems. *Proceedings of the 6th International Conference on Genetic Algorithms.* pp. 506-511.

Kobayashi, S., Ono, I. & Yamamura, M. 1996. A genetic algorithm for Job shop scheduling problems using job-based order crossover. *Proc. IEEE on Genetic Algorithms.* pp. 547-552.

Masters, T. 1993. *Practical neural network recipes in C++*. Academic Press, Inc.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. Teller, E. 1953. Equation of state calculations by fast computing machines. *Journal of Chemical Physics,* Vol 21, No 6; pp. 1087-1092.

Pinedo, M. & Chao, X. 1999. *Operations scheduling: With applications in manufacturing and services.* Boston: Irwin/McGraw-Hill.

Ritchie, E. & Tsado, A. K. 1986. A review of lot-sizing techniques for deterministic time-varying demand. *Production and Inventory Management.* pp: 65-79.

Silver, E. A. & Meal, H. C. 1973. Heuristic for selecting lot size requirements for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. *Production and Inventory Management* 14(2): 64-74.

Silver, E. A. & Peterson, R. 1985. *Decision systems for inventory management and production planning.* Second Edition*.* John Wiley & Sons.

Spears, W. M. 1998. The role of mutation and recombination in evolutionary algorithms. Ph. D. Dissertation, Fairfax, VA: George Mason University.

Tersine, R. J. 1994. *Principles of inventory and materials management.* 4th ed. New Jersey: Prentice-Hall, Inc.

Wagner, H. M., Whitin, T. M. 1958, Dynamic version of the economic lot size model. *Management Science* 5: 89-96.

Nasaruddin Zenon, Rosmah Ali and Ab Rahman Ahmad
Department of Industrial Computing,
The Faculty of Computer Science and Information System,
Universiti Teknologi Malaysia,
81310 Skudai, Johor.
rosmah@fsksm.utm.my
ahmadar@utm.my